Task Modeling and Specification for Modular Sensory Based Human–Machine Cooperative Systems

D. Kragic
Centre for Autonomous Systems
Royal Institute of Technology, Stockholm, Sweden
danik@nada.kth.se

G.D. Hager
Department of Computer Science
Johns Hopkins University, Baltimore, MD 21218
hager@cs.jhu.edu

Abstract— **This paper is directed towards developing Human-Machine Cooperative Systems (HCMS) for augmented surgical manipulation tasks. These tasks are commonly repetitive, sequential, and consist of simple steps. The transitions between these steps can be driven either by the surgeon's input or sensory information. Consequently, complex tasks can be effectively modeled using a set of basic primitives, where each primitive defines some basic type of motion (e.g. translational motion along a line, rotation about an axis, etc.). These steps can be "open-loop" (simply complying to user's demands) or "closed-loop, in which case external sensing is used to define a nominal reference trajectory.**

The particular research problem considered here is the development of a system that supports simple design of complex surgical procedures from a set of basic control primitives. The three system levels considered are: i) task graph generation which allows the user to easily design or model a task, ii) task graph execution which executes the task graph, and iii) at the lowest level, the specification of primitives which allows the user to easily specify new types of primitive motions. The system has been developed and validated using the JHU Steady Hand Robot as an experimental platform.

I. Introduction

Human-Machine Cooperative Systems combine human decision-making with sensory-robotic enhancement to accomplish complex tasks. The tasks performed in such a manner are typically difficult or impossible to perform without the human being physically present in the loop. There are two main communities where HMCS systems have been used extensively: i) telerobotics and ii) medical interventions. In the first case, HMCS have mainly been used in remote [1] and hazardous environments [2]. Microsurgery, which is also the key application considered in this paper, is one example of the latter, [4].

In terms of surgical applications, HMCS provide assistance and augmentation to surgeons during surgical procedures. Typically, the surgeon directly manipulates a tool which is also attached to a robot. Consequently, the behavior of the robot can be controlled either by i) complying to the user input, or ii) using the available sensory feedback to enhance the performance of the user. There are two operating scales commonly considered during procedures governing the type of feedback used. For cases of micro-scale manipulation (eye-surgery), visual feedback is commonly used. On the macro-scale, both force and visual feedback are facilitated. This implies that different levels of assistance/augmentation are needed depending on the current stage of the surgical procedure.

In the development of our system, the following issues were considered:
i) The system has to be *modular* - complex tasks should be defined using a set of basic control primitives. This allows the surgeon to model a variety of tasks using the existing architecture.
ii) The system should be able to provide the *appropriate* assistance to the user based on the task at hand. This is solved by allowing the surgeon to define different types of *constraints* depending on the operating scale.
iii) The system should be *flexible* to allow the users to easily change the existing model for the task they want to perform.
iv) The system should be *scalable* so that that the system structure executes both simple and complex tasks with same efficiency.
v) And finally, the system should have a *theoretical foundation* that allows for synthesis and verification.

The paper is organized as follows. In Section II a short overview of the related work is given. In Section III, the basic design of the system is presented. The notion of basic primitives required to design complex tasks is given in Section IV. Task specification approach is outlined Section V. The principles of task graph generation are presented in Section VI followed by the task execution strategy in Section VII. This is followed by an example in Section VIII.Finally, Section IX provides short summary and outlines avenues for future research.

II. Background and Related Work

In general, robots have no skills and require detailed instructions to complete high-level plans. Skills represent higher-level programming primitives which are task-relevant. In terms of HMCS, these primitives can be sensor and/or user guided. In other words, sensors should provide measurements for task-driven control. Closed–loop visual servoing can be used to achieve more stringent initial positioning preconditions compared to force based primitives. For medical applications, the system usually has some form of a model of the task being performed (or, at the far end, should be even able to build one). This is due to the fact that surgical procedures are repetitive and sequential. Therefore, similar to the Task Control architecture (TCA) [3], we can use a centralized process control model with a single supervisory module.

The work pursed in this paper builds upon the research presented in [5] and [6] . The former studies augmented surgical manipulation tasks in order to create an environment that makes it possible to easily and safely specify a set of control primitives (basic control modules) necessary for a number of different surgical procedures. The latter is concerned with the issue of providing the appropriate assistance to the users by recognizing their actions using continuous Hidden Markov Models.

Based on these, we are interested in building a system that provides an optimized augmentation based on the context of the task. Compared to the approach presented in [6], we propose an approach to task level modeling considering both unconstrained and constrained robot motion. By using a predefined modeling schema and a number of basic steps, it is easy for the user to model a variety of complex tasks. By using the proposed framework, the ideas in [5] can now easily be integrated in the system to build graphs for new tasks.

III. System Design

For each primitive, there are different requirements on the robot's behavior. By different behaviors of the robot, we consider translational and/or rotational motion, motion of a specific joint, etc. For each behavior, there may be safety constraints such as upper bound on the velocity, predefined level of stiffness/compliance based on the precision level, etc. An important issue in surgical applications is error recovery. In the current system, each state is therefore "glued" to an error state that can handle different types of error (discussed in more detail in Section IV).

As an example, retinal vein cannulation [7] involves *positioning* and *orienting* of a needle to the vicinity of the vein, *inserting* it when appropriate until contact is made. On contact, *puncturing* is performed, after which the needle can be safely *withdrawn*. These parts can be represented as *states* in a task graph, with *transitions* as connections between them, see Figure 1.

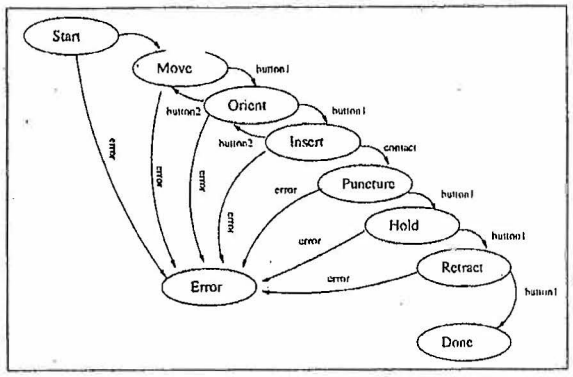

Fig. 1. An example of a basic task graph for vein cannulation.

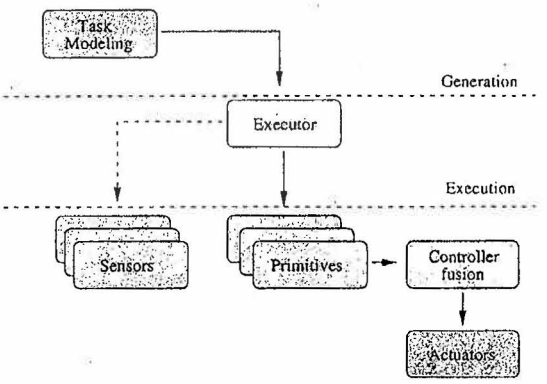

Fig. 2. The system architecture.

The current system is composed of three levels, see Figure 2:

i) Task graph modeling and generation allows the user to design and generate a graph for the task he/she wants to perform.

ii) Task graph execution - given the task graph, the system automatically initiates all the basic processes required to execute the task.

iii) A set of basic primitives is implemented and used during the execution of the first two levels. These primitives are the ones commonly used in surgical procedures. At this level, it is also easy to implement new primitives which are automatically detected by the previous two levels.

The system is currently tested using the JHU Steady-Hand robot [14], see Figure 3. The Steady-Hand is a 7DOF manipulator with XYZ translation at the base for coarse positioning, two rotational degrees of freedom at the shoulder, instrument insertion and rotation stages. There is a force sensing handle at the end-effector used to sense users' forces. Tools are mounted at the endpoint and

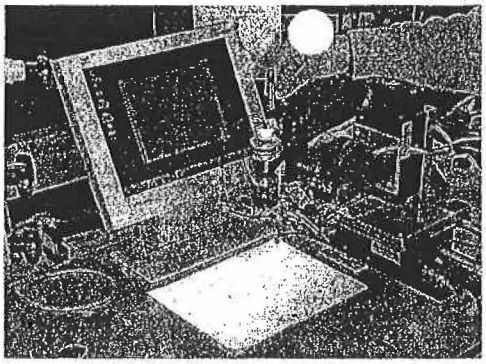

Fig. 3. The experimental setup of the JHU Steady Hand Robot.

"manipulated" by the operator holding the force handle. The robot responds to the applied force allowing the direct control for the operator. The overall positioning accuracy of the robot is less than 10 micrometers.

IV. Basic Primitives

The instructions to the low–level motion controller of the robot are passed from the basic primitives. The primitives have a common interface with functions such as Start(), Run() and Stop(). The core functions are defined in a base class, so if the user wants to design a new module, all the basic functionality is inherited from the base class and only the module specific parts have to be implemented. Basically, these are the steps required:

- Assign a name to the class.
- Initialize the necessary variables in the Start().
- Implement the control loop in Run().
- Clean up in Stop().

A. Virtual Fixtures as a Control Law

In our implementation, "virtual fixtures" provide cooperative control of the robot manipulator by "stiffening" a hand-held guidance mechanism against certain directions of motion or forbidden regions of the workspace. It has been indicated by studies on virtual fixtures for teleoperation that user performance can increase as much as 70% with the fixture based guidance [15]. In our framework, virtual fixtures are used to implement basic control blocks for the system used on JHU SHR.

In what follows, we model the robot as a purely kinematic Cartesian device with tool tip position $\mathbf{x} \in SE(3)$ and a control input that is endpoint velocity $\mathbf{v} = \dot{\mathbf{x}} \in \mathcal{R}^6$, all expressed in the robot base frame. The robot is guided by applying forces and torques $\mathbf{f} \in \mathcal{R}^6$ on the manipulator handle, likewise expressed in robot base coordinates.

The guidance for JHU SHR is defined geometrically by identifying a space of "preferred" directions of motion. Let us assume that we are given a $6 \times n$ time-varying matrix $D = D(t)$, $n < 6$, representing the instantaneous preferred directions of motion. Here, if n is 1, the preferred direction is along a curve in SE(3); if n is 2 the preferred directions span a surface, and so forth.

As an example, in Figure 1 the *Move* primitive would allow only the Cartesian base motion to move the remote center of motion (RCM) to the vicinity of the object to be manipulated. This is opposed to the *Orient* which uses only rotational and end-effector joints which keeps the RCM fixed for fine manipulation motions. The *Insert* primitive would allow only the motion along the current tool axis.

We define two projection operators, the span and the kernel of the column space, as

$$\text{Span}(D) \equiv \quad [D] = D(D'D)^{+}D' \tag{1}$$

$$\text{Ker}(D) \equiv \quad \langle D \rangle = I - [D] \tag{2}$$

where $^{+}$ denotes pseudo-inverse for the case where D is (column) rank deficient.

By decomposing the input force vector, $\mathbf{f}$, into two components

$$\mathbf{f}_D \equiv \quad [D]\mathbf{f} \tag{3}$$

$$\mathbf{f}_\tau \equiv \quad \mathbf{f} - \mathbf{f}_D = \langle D \rangle \mathbf{f} \tag{4}$$

and introducing a new admittance ratio $k_\tau \in [0, 1]$ that attenuates the non-preferred component of the force input, we arrive at an admittance control of the form

$$\mathbf{v} = \quad k(\mathbf{f}_D + k_\tau \mathbf{f}_\tau) \tag{5}$$

$$= \quad k([D] + k_\tau \langle D \rangle)\mathbf{f} \tag{6}$$

Thus, the final control law is in the general form of an admittance control with a time-varying gain matrix determined by $D(t)$. By choosing k, we control the overall admittance of the system. Choosing k_τ low imposes the additional constraint that the robot is stiffer in the non-preferred directions of motion. As noted above, we refer to the case of $k_\tau = 0$ as a *hard virtual fixture*, since it is not possible to move in any direction other than the preferred direction. All other cases will be referred to as *soft virtual fixtures*. In the case $k_\tau = 1$, we have an isotropic admittance.

As it will be shown in Section VI, during the task graph generation, the user can specify what type of motion is desired for the current primitive. Since all the primitives are added to the system dynamically at the beginning of the task execution, the system does not have to be compiled each time a new primitive is added.

B. Vision Based Virtual Fixtures as a Control Law

Some of the basic primitives incorporate additional visual feedback in the control loop. This way, the motion of the robot can additionally be constrained to a *reference*

direction fixture, [12]. In terms o sually guided control [8], a number of basic primitives which enforce kinematic constraints can be defined. To obtain visual measurements, the XVision system, [13] is used.

What we have presented in the previous section directly supports motion in a subspace, but it does not allow us to define a fixed desired motion trajectory. If $\mathbf{u} = f(\mathbf{x}, S)$ is the signed distance of the tool tip from a surface S, we can define a new preferred direction as

$$D_c(\mathbf{x}) = [(1-k_d)[\mathbf{D}]\mathbf{f}/\|\mathbf{f}\| + k_d\langle\mathbf{D}\rangle\mathbf{u}] \quad (7)$$

$$\text{with} \quad 0 < k_d < 1, \quad (8)$$

combining the two vectors that encode motion in the preferred direction and correcting the tool tip back to S. The constant k_d governs how quickly the tool is moved toward the reference surface. One minor issue here is that the division by $\|\mathbf{f}\|$ is undefined when no user force is present. As projection is invariant to scale, we write (7)

$$D_c(\mathbf{x}) = (1-k_d)[\mathbf{D}]\mathbf{f} + k_d\|\mathbf{f}\|\langle\mathbf{D}\rangle\mathbf{u} \quad (9)$$

$$\text{with} \quad 0 < k_d < 1 \quad (10)$$

and apply (5) with $D = D_c$.

One potential problem with this control law is that when the user applies no force, there is no virtual fixture because there is no defined preferred direction. Thus, there is a discontinuity at the origin. However, in practice the resolution of any force sensing device is usually well below the numerical resolution of the underlying computational hardware computing the pseudo-inverse, so the user will never experience this discontinuity.

As an example, for a line following primitive which may represent a vessel, a *preferred motion* will be along the direction of the line. During the task execution, which may be to perform a cut along the vessel, whenever the user deviates form the line, an error vector is defined based on the distance to the line, [12].To bring the surgeon back to the line, the preferred direction can now be defined perpendicular to the line until the error is equal to zero.

C. Error Handling

For surgical applications, it is of inevitable importance for the system to handle different types of errors. The error can be reported either by the system or by the user by pressing a button on a keyboard. When in the error state, the surgeon can i) terminate the execution of the task, ii) return back to the state from which the error was reported, or iii) choose some other state and continue the task.

V. Task Specification

Our system provides the user with a plethora of basic primitives which can be easily combined to perform complex tasks. A her important feature of the system is also the ability ω impose some basic requirements on the design of a task. For this reason, we have chosen to use the *XML Schema Definition Language (XSD)*, [11] based representation for a general class of procedures as follows:

```
<xs:element name="procedures">
    <xs:complexType><xs:sequence>
        <xs:element ref="event" minOccurs="1"
                maxOccurs="unbounded"/>
       <xs:element ref="basisVector" minOccurs="0"
                       maxOccurs="6"/>
        <xs:element ref="procedure" minOccurs="1"
                maxOccurs="unbounded"/>
-------------------------------------------------------
 <xs:element name="procedure">
    <xs:complexType> <xs:sequence>
        <xs:element name="description" type="xs:string"
    minOccurs="0" maxOccurs="1"/>
        <xs:element ref="state" minOccurs="1"
    maxOccurs="unbounded"/>
      <xs:attribute name="name" type="xs:string"
    use="required"/>
-------------------------------------------------------
<xs:element name="state">
    <xs:complexType><xs:sequence>
        <xs:element name="description"  type="xs:string"
            minOccurs="0" maxOccurs="1"/>
        <xs:element ref="transition"
            minOccurs="1"
            maxOccurs="unbounded"/>
        <xs:element ref="constraint"
            minOccurs="0"
            maxOccurs="unbounded"/>
      <xs:attribute name="name"
                    type="xs:string"
    use="required"/>
-------------------------------------------------------
<xs:element name="event">
    <xs:complexType><xs:sequence>
        <xs:element name="description" type="xs:string"
            minOccurs="0" maxOccurs="1"/>
      <xs:attribute name="type"  type="xs:ID"
    use="required"/>
-------------------------------------------------------
  <xs:element name="constraint">
    <xs:complexType> <xs:sequence>
        <xs:element name="description" type="xs:string"
      minOccurs="0" maxOccurs="1"/>
      <xs:attribute name="constr" type="xs:IDREF"
      use="required"/>
      <xs:attribute name="weight" type="xs:double"
      use="required"/>
```

In our system, we assume that surgical interventions can be modeled by a set of events, basis vectors and a procedure with a sequence of states. Events represent links or triggers between the states and are either sensory based (e.g. *contact detected*) or induced by the user (e.g. *button pressed*). Basis vectors span the task space of the robot. Using different types of operators on the basis vectors, we can easily define subspaces for preferred robot motion. States are represented by a set of transitions and constraints. Transitions are pairs (event, newState), i.e. for each event there is a newState defined.

There may be a number of constraints defined for a certain state. Those are directly connected with the behavior of the system through the control algorithm. As examples of constraints, there may be a value which defines the level

of compliance/stiffness of the rc definition of virtual fixtures, etc. According to this specification schema, the Extensible Markup Language (XML), [11] can now be used for task graph generation.

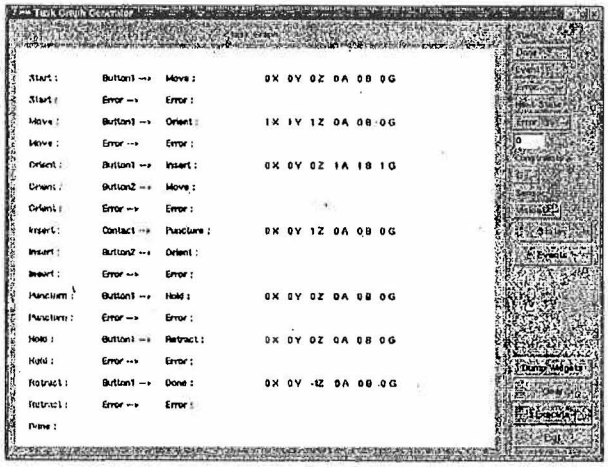

Fig. 4. Graph generation using a simple GUI. For each state, a number of events are specified. In addition, the user can define a set of preferred directions and their magnitude (X, Y, Z represent the three translational degrees of freedom and A, B and G represent the three axis of rotation). In addition, for each primitive the type of sensor can be defined.

VI. Task Graph Generation

There are three ways of generating a task graph:
i) **GUI** - using a graphical interface that relies on the system structure and the specification file, see Figure 4.
ii) **Learning** - in [5], it was shown how a simple language for modeling of multiple human–machine cooperative tasks can be designed. The underlying assumption was that a task is composed of *gestemes* - small component actions which are supported by the system. These are then be combined to generate task graphs on–line by recognizing users' actions.
iii) Directly specifying the XML file.

VII. Task Graph Execution

The task graph manager, see Figure 2, configures the *procedure* with its underlying structure to solve the task given the task graph. When invoking the control system, the task graph manager is initiated with command line parameters specifying a text file produced during task graph generation. The text file is interpreted so that the tree structure is built. After the task manager has received a confirmation that the whole tree is successfully configured, the system can be run. The system is event based and the task manager either decides which method to run or stops the execution in case of an error.

VI Example Scenario

Let us now study the execution of the vein cannulation procedure as shown in Figure 1. The following primitives are used to model the task:
MOVE: move the instrument from a starting position to some position close to the surface,
ORIENT: align the instrument parallel/perpendicular to the surface,
INSERT: move the instrument tip along current tool axis through until touching the opposite inner cylinder wall,
PUNCTURE and HOLD: prepare and perform puncturing,
RETRACT: carefully withdraw the instrument tip in the direction of the tool axis,
DONE: move instrument roughly back to the starting position/away from the surface and report that the task was successfully performed,
ERROR: if an error is reported, either i) exit or ii) continue with the execution allowing the user to choose the next state.

Now, the pseudo XML representation is as follows:

```
<event type="btn1"
<event type="btn2"
<event type="err"
<event type="cnt"
-----------------------------------------------
<basisVector opcode="bvX">
<vector> 1.0 0.0 0.0 0.0 0.0 0.0
<basisVector opcode="bvY">
<vector> 0.0 1.0 0.0 0.0 0.0 0.0
<basisVector opcode="bvZ">
<vector> 0.0 0.0 1.0 0.0 0.0 0.0
<basisVector opcode="bvA">
<vector> 0.0 0.0 0.0 1.0 0.0 0.0
<basisVector opcode="bvB">
<vector> 0.0 0.0 0.0 0.0 1.0 0.0
<basisVector opcode="bvG">
<vector> 0.0 0.0 0.0 0.0 0.0 1.0
-----------------------------------------------
<state name="MoveTo">
  <description> Move the tool closer.
  <transition event="btn1" newState="Orient"
  <transition event="err" newState="Error"
  <constraint constr="bvX" weight="1.0"
  <constraint constr="bvY" weight="1.0"
  <constraint constr="bvZ" weight="1.0"
-----------------------------------------------
<state name="Orient">
  <description> Orient the tool.
  <transition event="btn1" newState="Insert"
  <transition event="btn2" newState="moveTo"
  <transition event="err" newState="Error"
  <constraint constr="bvA" weight="1.0"
  <constraint constr="bvB" weight="1.0"
  <constraint constr="bvG" weight="1.0"
-----------------------------------------------
<state name="Insert">
  <description> Insert the tool.
  <transition event="btn2" newState="Orient"
  <transition event="cnt" newState="Puncture"
  <transition event="err" newState="Retract"
  <constraint constr="bvZ" weight="1.0"
-----------------------------------------------
<state name="Retract">
  <description> Move the tool away.
  <transition event="btn1" newState="Done"
  <transition event="err" newState="Error"
  <constraint constr="bvZ" weight="-1.0"
```

The events are at this stage ıulated by the user pushing one of the keyboard buttons. The task space of the robot is defined by a set of basis vectors. Theses basis vectors are then used in each of the states to define the preferred motion for the robot. At the run time, each of states sends the defined set of preferred motions to the low level Steady Hand controller which implements simple virtual fixture control law as described in Section IV-A.

IX. Conclusion

In this paper, a Human-Machine Cooperative System for augmented surgical manipulation tasks was presented. The current system consists of three levels: i) task graph modeling and generation, ii) task graph execution and iii) low–level implementation of control primitives. The motivation for such a design is that the complex surgical tasks are commonly repetitive and sequential in nature consisting of simple steps. In the current system, the transitions between these steps are driven either by surgeon's input or sensory information. Consequently, complex tasks are modeled using a set of basic steps or primitives where each primitive defines some basic type of motion (e.g. translational motion along a line, rotation about an axis, etc.). In terms of control, the system simply complies to the users input where a number of different constraints (virtual fixtures) can be imposed.

The system is currently validated using the JHU Steady Hand Robot as an experimental platform. The example task presented above is analogous to many minimally invasive or constrained motion tasks. For medical applications, there are currently no systems that integrate both the visual and users input to define control input to the robot. We believe that our approach, which uses virtual fixtures to define a set of preferred robot motions, offers a powerful theoretical ground that allows for easy verification of control policies. Our future research will consider further experimentation with skilled users for applications using clinical conditions.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-0205318 and EEC-9731478. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. This research has partially been sponsored by the Swedish Foundation for Strategic Research through the Centre for Autonomous Systems. The funding is gratefully acknowledged.

X. References

[1] T.B. Sheridan, "Telerobotics, Automation and Human Supervisory Control", *MIT Press: Cambridge MA*, 1992.

[2] T. Debus, P. ɔont, R. Howe, "Automatic identification of local geometric properties during teleoperation", *IEEE International Conference on Robotics and Automation, ICRA2000*, pp. 3428-3434, vol 4, 2000

[3] R.Simmons, R.Goodwin, C.Fedor and J.Basista, "Task control architecture programmer's guide to version 8.0", http://www.cs.cmu.edu/~ reids, May 1997.

[4] R. Kumar, P.Berkelman, P. Gupta, A. Barnes, P. Jensen, L. Withcomb, R:T.aylor, "Preliminary experiments in cooperative human/robot force control for robot assisted micrsurgical manipulation", *IEEE International Conference on Robotics and Automation, ICRA2000*, pp. 610-617, vol 1, 2000

[5] C.S. Hundtofte, G.D. Hager, A.M. Okamura, "Building a task language for segmentation and recognition of user input to cooperative manipulation systems", *10th INternational Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, (2002 IEEE Virtual Reality Conference)*, pp. 225-230, 2000.

[6] R. Kumar, "An Augmented Steady Hand System for Precise Micromanipulation", *Ph.D. Thesis*, Department of Computer Science, The Johns Hopkins University, 2001.

[7] J.N. Weiss, "Injection of tissue plasminogen activator into a branch retinal vein in eyes with central retinal vein occlusion", *Opthalmology*, **108**(12), pp. 2249-2257, 2001.

[8] S. Hutchinson, G.D.Hager and P.Corke "A tutorial on visual servo control", *IEEE Transactions on Robotics and Automation* **12**(5), pp. 651–670, 1996.

[9] D. Kragic. "Visual Servoing for Manipulation: Robustness and Integration Issues", *PhD thesis*, Royal Institute of Technology, Stockholm, Sweden, 2001.

[10] L. Petersson, D. Austin, and H.I. Christensen. "DCA: A Distributed Control Architecture for Robotics", *IEEE International Conference on Intelligent Robots and Systems, IROS2001*, pp.2361–2368, vol 3, 2001

[11] www.xml.org

[12] G.D.Hager, "Human-machine cooperative manipulation with vision-based motion constraints", *In Proc. Workshop on Visual Servoing, with IEEE IROS*, 2002.

[13] G.Hager and K.Toyama, "The XVision system: A general–purpose substrate for portable real–time vision applications", *Computer Vision and Image Understanding* vol.69(1),pp. 23-37, 1996.

[14] R.H.Taylor, P.S.Jensen, L.L.Whitcomb, A.C.Barnes, R.Kumar, D.Stoianovici, P.Gupta, Z.Wang, E. Jr.deJuan and L.Kavoussi, "A Staedy-Hand Robotic Syatem for Microsurgical Augmentation", *IJRR*, vol.18, pp.1201-1210, 1999.

[15] L. Rosenberg, Virtual Fixtures. PhD Thesis, Dept. of Mech. Eng., Stanford University, 1994.